CENG 3420
Computer Organization & Design

Lecture 17: Cache-3 Examples

Bei Yu
CSE Department, CUHK
byu@cse.cuhk.edu.hk

(Textbook: Chapters 5.3–5.4)

Spring 2022

Overview

1. Example 1
2. Example 2
3. Example 3

Example 1

Cache Example

J for loop

A[0][0]
A[1][0]
A[2][0]
…
A[9][0]

i for loop

A[9][0]
A[8][0]
..
A[0][0]

```
short  A[10][4];
int    sum = 0;
int    j, i;
double mean;

// forward loop
for (j = 0; j <= 9; j++)
    sum += A[j][0];

mean = sum / 10.0;

// backward loop
for (i = 9; i >= 0; i--)
    A[i][0] = A[i][0]/mean;
```

- Assume separate instruction and data caches
- So we consider only the data
- Cache has space for 8 blocks

(so in direct mapping, we need 3 bits for block ID)

- A block contains one word (byte)
- `A[10][4]` is an array of words located at `7A00-7A27` in row-major order

Cache Example

Memory word address in hex	Memory word address in binary	Array Contents (40 elements)
(7A00)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	A[0][0]
(7A01)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1	A[0][1]
(7A02)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0	A[0][2]
(7A03)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1	A[0][3]
(7A04)	0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0	A[1][0]
⋮	⋮	⋮
(7A24)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0	A[9][0]
(7A25)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 1	A[9][1]
(7A26)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0	A[9][2]
(7A27)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 1	A[9][3]

Tag for Direct Mapped (first 13 bits)

8 blocks in cache, 3 bits encodes cache block number

Tag for Set-Associative (first 15 bits)

4 blocks/ set, 2 cache sets, 1 bit encodes cache set number

Tag for Associative (all 16 bits)

To simplify discussion: 16-bit word (byte) address; i.e. 1 word = 1 byte.

Direct Mapping

- Least significant 3-bits of address determine location
- No replacement algorithm is needed in Direct Mapping
- When `i == 9` and `i == 8`, get a cache hit (2 hits in total)
- Only 2 out of the 8 cache positions used
- Very inefficient cache utilization

		Content of data cache after loop pass: (time line)																			
		j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	i = 9	i = 8	i = 7	i = 6	i = 5	i = 4	i = 3	i = 2	i = 1	i = 0
Cache Block number	0	A[0][0]	A[0][0]	A[2][0]	A[2][0]	A[4][0]	A[4][0]	A[6][0]	A[6][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[6][0]	A[6][0]	A[4][0]	A[4][0]	A[2][0]	A[2][0]	A[0][0]
	1																				
	2																				
	3																				
	4		A[1][0]	A[1][0]	A[3][0]	A[3][0]	A[5][0]	A[5][0]	A[7][0]	A[7][0]	A[9][0]	A[9][0]	A[9][0]	A[7][0]	A[7][0]	A[5][0]	A[5][0]	A[3][0]	A[3][0]	A[1][0]	A[1][0]
	5																				
	6																				
	7																				

Tags not shown but are needed.

Associative Mapping

- LRU replacement policy: get cache hits for $i = 9, 8, \ldots, 2$
- If `i` loop was a forward one, we would get no hits!

Content of data cache after loop pass: (time line)

Cache Block number	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	i = 9	i = 8	i = 7	i = 6	i = 5	i = 4	i = 3	i = 2	i = 1	i = 0
0	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[0][0]
1		A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[1][0]	A[1][0]
2			A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[2][0]
3				A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]
4					A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]
5						A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]
6							A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]
7								A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]

Tags not shown but are needed; LRU Counters not shown but are needed.

Set Associative Mapping

- Since all accessed blocks have even addresses (`7A00, 7A04, 7A08, ...`), only half of the cache is used, i.e. they all map to set 0
- LRU replacement policy: get hits for i = 9, 8, 7 and 6
- Random replacement would have better average performance
- If `i` loop was a forward one, we would get no hits!

Content of data cache after loop pass: (time line)

Set	Cache Block number	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	i = 9	i = 8	i = 7	i = 6	i = 5	i = 4	i = 3	i = 2	i = 1	i = 0
Set 0	0	A[0][0]	A[0][0]	A[0][0]	A[0][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[8][0]	A[4][0]	A[4][0]	A[4][0]	A[4][0]	A[0][0]
	1		A[1][0]	A[1][0]	A[1][0]	A[1][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[9][0]	A[5][0]	A[5][0]	A[5][0]	A[5][0]	A[1][0]	A[1][0]
	2			A[2][0]	A[2][0]	A[2][0]	A[2][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[6][0]	A[2][0]	A[2][0]	A[2][0]
	3				A[3][0]	A[3][0]	A[3][0]	A[3][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[7][0]	A[3][0]	A[3][0]	A[3][0]	A[3][0]
Set 1	4																				
	5																				
	6																				
	7																				

Tags not shown but are needed; LRU Counters not shown but are needed.

Comments on the Example

- In this example, Associative is best, then Set-Associative, lastly Direct Mapping.
- What are the advantages and disadvantages of each scheme?
- In practice,
  - Low hit rates like in the example is very rare.
  - Usually Set-Associative with LRU replacement scheme is used.
- Larger blocks and more blocks greatly improve cache hit rate, i.e. more cache memory

Example 2

Question:

How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Question:

How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Answer:

- In a 32-bit address CPU, 16 KiB is 4096 words.
- With a block size of 4 words, there are 1024 blocks.
- Each block has 4×32 or 128 bits of data plus a tag, which is $(32 - 10 - 2 - 2) = 18$ bits, plus a valid bit.
- Thus, the total cache size is $2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147$ bits.
- For this cache, the total number of bits in the cache is about 1.15 times as many as needed just for the storage of the data.

1.15 = 147 / (4 x 32)

Example 3

Question

We have designed a 64-bit address direct-mapped cache, and the bits of address used to access the cache are as shown below:

Table: Bits of the address to use in accessing the cache

Tag	Index	Offset
63-10	9-5	4-0

1. What is the block size of the cache in words?
2. Find the ratio between total bits required for such a cache design implementation over the data storage bits.
3. Beginning from power on, the following byte-addressed cache references are recorded as shown below.

Table: Recored byte-addressed cache references

Hex	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
Dec	0	4	16	132	232	160	1024	30	140	3100	180	2180

Find the hit ratio.

1. Each cache block consists of four 8-byte words. The total offset is 5 bits. Three of those 5 bits is the word offset (the offset into an 8-byte word). The remaining two bits are the block offset. Two bits allows us to enumerate $2^2 = 4$ words.

2. The ratio is 1.21. The cache stores a total of $32\text{lines} \times 4\text{words/block} \times 8\text{bytes/word} = 1024\text{bytes} = 8192\text{bits}$. In addition to the data, each line contains 54 tag bits and 1 valid bit. Thus, the total bits required is $8192 + 54 \times 32 + 1 \times 32 = 9952 \quad \text{bits}$.

3. The hit ratio is $\frac{4}{12} = 33\%$

Byte Address	Binary Address	Tag	Index	Offset	Hit/Miss	Bytes Replaced
0x00	0000 0000 0000	0x0	0x00	0x00	M	
0x04	0000 0000 0100	0x0	0x00	0x04	H	
0x10	0000 0001 0000	0x0	0x00	0x10	H	
0x84	0000 1000 0100	0x0	0x04	0x04	M	
0xe8	0000 1110 1000	0x0	0x07	0x08	M	
0xa0	0000 1010 0000	0x0	0x05	0x00	M	
0x400	0100 0000 0000	0x1	0x00	0x00	M	0x00-0x1F
0x1e	0000 0001 1110	0x0	0x00	0x1e	M	0x400-0x41F
0x8c	0000 1000 1100	0x0	0x04	0x0c	H	
0xc1c	1100 0001 1100	0x3	0x00	0x1c	M	0x00-0x1F
0xb4	0000 1011 0100	0x0	0x05	0x14	H	
0x884	1000 1000 0100	0x2	0x04	0x04	M	0x80-0x9f